SPIM: A Pipelined 64 X 64 bit Iterative Multiplier

Mark R. Santoro
Mark A. Horowitz
Center for Integrated Systems
Stanford University
Stanford, CA. 94305
(415)725-3707

Abstract

A 64 by 64 bit iterating multiplier, SPIM (Stanford Pipelined Iterative Multiplier) is presented. The pipelined array consists of a small tree of 4:2 adders. The 4:2 tree is better suited than a Wallace tree for a VLSI implementation because it is a more regular structure. A 4:2 carry save accumulator at the bottom of the array is used to iteratively accumulate partial products, allowing a partial array to be used, which reduces area. SPIM was fabricated in a 1.6μm CMOS process. It has a core size of 3.8 X 6.5mm and contains 41 thousand transistors. The on chip clock generator runs at an internal clock frequency of 85MHz. The latency for a 64 X 64 bit fractional multiply is under 120ns, with a pipeline rate of one multiply every 47ns.

SPIM: A Pipelined 64 X 64 bit Iterative Multiplier

Mark R. Santoro
Mark A. Horowitz
Center for Integrated Systems
Stanford University
Stanford, CA. 94305

I. Introduction

The demand for high performance floating point coprocessors has created a need for high-speed, small-area multipliers. Applications such as DSP, graphics, and on chip multipliers for processors require fast area efficient multipliers. Conventional array multipliers achieve high performance but require large amounts of silicon, while shift and add multipliers require less hardware but have low performance. Tree structures achieve even higher performance than conventional arrays but require still more area.

The goal of this project was to develop a multiplier architecture which was faster and more area efficient than a conventional array. As a test vehicle for the new architecture a structure capable of performing the mantissa portion of a double extended precision (80 bit) floating point multiply was chosen. The multiplier core should be small enough such that an entire floating point co-processor, including a floating point multiplier, divider, ALU, and register file, could be fabricated on a single chip. A core size of less than 25mm^2 was determined to be acceptable. This paper presents a 64 by 64 bit pipelined array iteratively accumulating multiplier, SPIM (Stanford Pipelined Iterative Multiplier), which

can provide over twice the performance of a comparable conventional full array at 1/4 of the silicon area.

II. Architectural Overview

Conventional array multipliers consist of rows of carry save adders (CSA) where each row of carry save adders sums up one additional partial product (see Figure 1).[1] Since intermediate partial products are kept in carry save form there is no carry propagate, so the delay is only dependent upon the depth of the array and is independent of the partial product width. Although arrays are fast, they require large amounts of hardware which is used inefficiently. As the sum is propagated down through the array, each row of carry save adders is used only once. Most of the hardware is doing no useful work at any given time. Pipelining can be used to increase hardware utilization by overlapping several calculations. Pipelining greatly increases throughput, but the added latches increase both the required hardware, and the latency.

Since full arrays tend to be quite large when multiplying double or extended precision numbers, chip designers have used partial arrays and iterated using the system clock. This structure has the benefit of reducing the hardware by increasing utilization. At the limit, an iterative structure would have one row of carry save adders and a latch. Figure 2 shows a minimal iterative structure. Clearly, this structure requires the least amount of hardware and has the highest utilization since each CSA is used every cycle. An important observation is that iterative structures can be made fast if the latch delays are small, and the clock is matched to the combinational delay of the carry save

[1]Carry save adders are also often referred to as full adders or 3:2 adders.

adders. If both of these conditions are met the iterative structure approaches the same throughput and latency as the full array. This structure does, however, require very fast clocks. For a 2μm process clocks may be in the 100MHz range. A few companies use iterative structures in their new high performance floating point processors [5].

In an attempt to increase performance of the minimal iterative structure additional rows of carry save adders could be added, resulting in a bigger array. For example, addition of a row of CSA cells to the minimal structure would yield a partial array with two rows of Carry Save Adders. This structure provides two advantages over the single row of CSA cells: it reduces the required clock frequency, and requires only half as many latch delays.[2] One should note, however, that although we doubled the number of carry save adders, the latency was only reduced by halving the number of latch delays. The number of CSA delays remains the same. Increasing the depth of the partial array by simply adding additional rows of carry save adders in a conventional structure yields only a slight performance increase. This small reduction in latency is the result of reducing the number of latches.

To increase the performance of this iterative structure we must make the CSA cells fast and, more importantly, decrease the number of series adds required to generate the product. Two well known methods for the latter are Booth encoding and tree structures [2][9]. Modified Booth encoding, which halves the number of series adds required, is used on most modern floating point chips,

[2]In fact one rarely finds a multiplier array that consists of only a single row of carry save adders. The latch overhead in this structure is extremely high.

including SPIM [7][8]. Tree structures reduce partial products much faster than conventional methods, requiring only order logN CSA delays to reduce N partial products (see Figure 3). Though trees are faster than conventional arrays, like conventional arrays they still require one row of CSA cells for each partial product to be retired. Unfortunately, tree structures are notoriously hard to lay out, and require large wiring channels. The additional wiring makes full trees even larger than full arrays. This has caused designers to look at permutations of the basic tree structure [1][11]. Unbalanced or modified trees make a compromise between conventional full arrays and full tree structures. They reduce the routing required of full trees but still require one row of carry save adders for each partial product. Ideally one would want the speed benefits of the tree in a smaller and more regular structure. Since high performance was a prerequisite for SPIM a tree structure was used. This left two problems. The first, was the irregularity of commonly used tree structures. The second problem was the large size of the trees.

Wallace [9], Dadda [4], and most other multiplier trees use a carry save adder as the basic building block. The carry save adder takes 3 inputs of the same weight and produces 2 outputs. This 3:2 nature makes it impossible to build a completely regular tree structure using the CSA as the basic building block. A binary tree has a symmetric and regular structure. In fact, any basic building block which reduces products by a factor of two will yield a more regular tree than a 3:2 tree. Since a more regular tree structure was needed the solution was to introduce a new building block: the 4:2 adder, which reduces 4 partial products of the same weight to 2 bits. Figure 4 is a block diagram of the 4:2 adder. The truth table for the 4:2 adder is shown in Table 1. Notice that the 4:2

adder actually has 5 inputs and 3 outputs. It is different from a 5:3 counter which takes in 5 inputs of the same weight and produces 3 outputs of different weights. The sum output of the 4:2 has weight 1 while the Carry and Cout both have the same weight of 2. In addition, the 4:2 is not a simple counter as the Cout output must NOT be a function of the Cin input or a ripple carry could occur. As for the name, 4:2 refers to the number of inputs from one level of a tree and the number of outputs produced at the next lower level. That is, for every 4 inputs taken in at one level, two outputs are produced at the next lower level. This is analogous to the binary tree in which for every 2 inputs 1 output is produced at the next lower level. The 4:2 adder can be implemented directly from the truth table, or with two carry save add (CSA) cells as in Figure 5.[3]

A 4:2 tree will reduce partial products at a rate of $\log_2(N/2)$ whereas a Wallace tree requires $\log_{1.5}(N/2)$; where N is the number of inputs to be reduced. Though the 4:2 tree might appear faster than the Wallace tree, the basic 4:2 cell is more complex so the speed is comparable. The 4:2 structure does however yield a tree which is much more regular. In addition the 4:2 adder has the advantage that two Carry Save Adders are in each pipe in place of one. This reduces both the required clock frequency and the latch overhead.

To overcome the size problem SPIM uses a partial 4:2 tree, and then iteratively accumulates partial products in a carry save accumulator to complete the computation. The carry save accumulator is simply a 4:2 adder with two of the

[3]SPIM implemented the 4:2 adder with two CSA cells because it permits a straight forward comparison with other architectures on the basis of CSA delays. By knowing the size and speed of the CSA cells in any technology a designer can predict the size and speed advantages of this method over that currently used.

inputs used to accumulate the previous outputs. The carry save accumulator is much faster than a carry propagate accumulator and requires only one additional pipe stage.

Figure 6 compares a single 4:2 adder with carry save accumulator, to a conventional partial piped array.[4] Both structures reduce 4 partial products per cycle. Notice, however, that the tree structure is clocked at almost twice the frequency of the partial piped array. It has only 2 CSA cells per pipe stage, whereas the partial piped array has 4. Consequently, the partial array would require 32 CSA delays to reduce 32 partial products where the tree structure would need only 18 CSA delays. Using the 4:2 adder with carry save accumulator is almost twice as fast as the partial piped array, while using roughly the same amount of hardware.

The 4:2 adder structure can be used to construct larger trees, further increasing performance. In Figure 7 we use the same 4:2 adder structure to form an 8 input tree. This allows us to reduce 8 partial products per cycle. Notice that we still pipeline the tree after every 2 carry save adds (each 4:2 adder). In contrast, if we clocked the tree every 4 carry save adds it would double the cycle time and only decrease the required number of cycles by one. The overall effect would be a much slower multiply.

[4]In figures 6, 7, and 9 the detailed routing has not been shown. Providing the exact detailed routing, as was done in figure 5, would provide more information; however, it would significantly complicate the figures and would tend to obscure their purpose, which is to show the data flow in terms of pipe stages and carry save add delays.

Figure 8 shows the size and speed advantages of different sized 4:2 trees with carry save accumulators vs. conventional partial arrays. This plot is a price/performance plot where the price is size and the performance is speed (latency = 1/speed). The plot assumes we are doing a 64 X 64 bit multiply. Booth encoding is used, thus we must retire 32 partial products. Size has been normalized such that 32 rows of CSA cells (a full array) has a size of 1 unit.[5] In the upper left corner is the structure using only 2 rows of CSA cells. In this case the tree and conventional structures are one and the same and can be seen as a partial array 2 rows deep, or as a 2 input partial tree. We can see that adding hardware to form larger partial arrays provides very little performance improvement. A full array is only 15% faster than the iterative structure using 2 rows of carry save adders. Adding hardware in a tree type structure however, dramatically improves performance. For example, using a 4 input tree, which uses 4 rows of carry save adders, is almost twice as fast as the 2 input tree. Using an 8 input tree is almost 3 times as fast as a 2 input tree and only 1/4 the size of the full array.

The latency of the multiplier is determined by the depth of the partial 4:2 tree and the fraction of the partial products compressed each cycle. The latency is equal to $\log_2(K/2) + (N/K)$ where N is the operand size and K is the partial tree size. If Booth encoding is used N would be one half the operand size since Booth encoding has already provided a factor of 2 compression. Startup times and pipe stages before the tree must also be taken into account when determining latency. We choose the 8 input piped tree with Booth encoding for

[5]Latency is in terms of CSA delays. We have assumed a latch is equivalent to 1/3 of a CSA delay in an attempt to take the latch delays into account. Size is the number of CSA cells used. It does not include the latch or wiring area.

SPIM, as we felt this provided best area speed tradeoff for our purpose. The number of cycles required to reduce 64 bits using Booth encoding and an 8 bit tree is:

$$\log_2(8/2) + (32/8) + \text{one cycle overhead} = 7 \text{ cycles}^6$$

III. SPIM Implementation

Figure 9 is a block diagram of the SPIM data path. The Booth encoders, which encode 16 bits per cycle, are to the left of the data path. The Booth encoded bits drive the Booth select muxes in the A and B block. The A and B block Booth select mux outputs drive an 8 input tree structure constructed of 4:2 adders which are found in the A, B, and C blocks. Each pipe stage uses one 4:2 adder which consists of two carry save adders. The D block is a carry save accumulator. It also contains a 16 bit hard wired right shift to align the partial sum from the previous cycle to the current partial sum to be accumulated.

Figure 10 is a die photograph of SPIM. The A block inputs are pre-shifted allowing the A block to be placed on top of B block. Using 4:2 adders in a partial tree allows the array to be efficiently routed, and laid out as a bit slice, thus making the SPIM array a very regular structure. Interestingly, the CSA cells occupy only 27% of the core area. The Booth select muxes used in the A and B blocks make these blocks three times as large as the C block. Each Booth mux with it's corresponding latch is larger than a single carry save adder. Also, due to the routing required for the 16 bit shift, the D block is twice as large as the C block. The array area can be split into four main components; routing,

[6]The one cycle overhead is used for the Booth select muxes.

CSA cells, muxes, and latches. The routing required 20% of the area, while the other 75% was equally split between the CSA cells, muxes, and latches.

The critical path in the SPIM data path is through the D block. The D block contains the slowest path because of the added routing at the output, and the additional control mux at its input. The input mux is needed to reset the carry save accumulator. It selects "0" to reset, or the previous shifted output when accumulating. The final critical path through the D block includes 2 CSA cells, a master slave latch, a control mux, and the drive across 16 bits (128μm) of routing.

IV. Clocking

The architecture of SPIM yields a very fast multiply; however, the speed at which the structure runs demands careful attention to clocking issues. Only two carry save adders (one 4:2 adder) are found in each pipe stage, yielding clock rates on the order of 100MHz. The typical system clock is not fast enough to be useful for this type of structure. To produce a clock of the desired frequency SPIM uses a controllable on chip clock generator. The clock is generated by a stoppable ring oscillator. The clock is started when a multiply is initiated, and stopped when the array portion of the multiply has been completed. The use of a stoppable clock provides two benefits. It prevents synchronization errors from occurring and it saves power as the entire array is powered down upon completing a multiply. The actual clock generator used on SPIM is shown in Figure 11. It has a digitally selectable feedback path which provides a programmable delay element for test purposes. This allows the clock frequency

to be tuned to the critical path delay. In addition, the clock generator has the ability to use an external test clock in place of the fast internally generated clock.

When a multiply signal has been received a small delay occurs while starting up the clocks. This delay comes from two sources. The first is the logic which decodes the run signal and starts up the ring oscillator. The second source of delay is from the long control and clock lines running across the array. They have large capacitive loads and require large buffer chains to drive them. The simulated delay of the buffer chain and associated logic is 6ns, almost half a clock cycle. Since the inputs are latched before the multiply is started, SPIM does the first Booth encode before the array clocks become active (cycle 0). Thus, the startup time is not wasted. After the clocks have been started SPIM requires seven clock cycles (cycles 1-7) to complete the array portion of a multiply.

The detailed timing is shown in Table 2. In the time before the clocks are started (cycle 0) the first 16 bits are Booth encoded. During cycle 1, the first 16 Booth-coded partial products from cycle 0 are latched at the input of the array. The next four cycles are needed to enter all 32 Booth-coded partial products into the array. Two additional cycles are needed to get the output through the C and D blocks. If a subsequent multiply were to follow it would have been started on cycle 4, giving a pipelined rate of 4 cycles per multiply. When the array portion of the multiply is complete the carry save result is latched, and the run signal is turned off. Since the final partial sum from the D block is latched into the carry propagate adder only every fourth cycle, several cycles are available to stop the clock without corrupting the result.

The clock generator is located in the lower left hand side of the die (see Figure 10). The clock signal runs up a set of matched buffers, along the side of the array, which are carefully tuned to minimize skew across the array. Wider than minimum metal lines are used on the master clock line to reduce the resistance of the clock line relative to the resistance of the driver. The clock and control lines driven from the matched buffers then run across the entire width of the array in metal.

V. Test Results

To accurately measure the internal clock frequency the clock was made available at an output allowing an oscilloscope to be attached. SPIM was then placed in continuous (loop) mode where the clock is kept running and multiplies are piped through at a rate of one multiply every 4 cycles. Since the clock is continuously running its frequency can be accurately determined.

Three components determine the actual performance of SPIM. The startup time, when the clocks are started and the first Booth encode takes place (cycle 0), the array time, which includes the time through the partial array plus the accumulation cycles (cycles 1-7), and the carry propagate addition time, when the final carry propagate addition converts the carry save form of the result from the accumulator to a simple binary representation. Due to limitations in our testing equipment only the array time could be accurately measured. Since the array time requires 7 cycles, and the array clock frequency was 85MHz the array time is simply 7 * (1/85MHz) = 82.4ns. The startup and cpadd times,

based upon simulations, were 6ns and 30ns respectively. In flowthrough mode the total latency is simply the sum of the startup time (6ns), the array time (82.4ns), and the cpadd time (30ns), for a total of 118.4ns. Thus SPIM has a total latency under 120ns. SPIM has a throughput of one multiply every 4 cycles or 4 * (1/85MHz) = 47ns, for a maximum pipelined rate in excess of 20 million 80 bit floating point multiplies per second.

The performance range of the parts tested was from 85.4MHz to 88.6MHz at a room temperature of 24.5 ^{o}C and a supply voltage of 4.9 volts. One of the parts was tested over a temperature range of 5 to 100 ^{o}C. At 5 ^{o}C it ran at 93.3MHz with speeds of 88.6MHz and 74.5MHz at 25 and 100 ^{o}C. The average power consumed at 85MHz was 72mA while an average of only 10mA was consumed in standby mode.

VI. Future Improvements

The Booth select muxes with their corresponding latches account for 38% of the array area. This was larger than expected. Though Booth encoding reduces the number of partial products by a factor of two, the same result could be achieved by adding one more level of 4:2 adders to the tree. Since much of the routing already exists for the Booth muxes, adding another level to the tree requires replacing each two Booth select muxes with a 4:2 adder and 4 AND gates (see Figure 12). Since the CSA cells are slightly larger than the Booth select muxes the array size will grow slightly, (by about 7%). However, if we take the whole picture into account, the core would remain about the same size, as we would no longer need the Booth encoders. Replacing the Booth

encoders and Booth select muxes with an additional level to the tree would also reduce the latency by one cycle from 7 cycles to 6. This occurs because the cycle required to Booth encode is now no longer needed. There are other advantages in addition to the increase in speed. Perhaps the greatest gain is the reduction in complexity. Both the Booth encoders and Booth select muxes are now unnecessary, thus the number of cells has been reduced. In addition, Booth encoding generates negative partial products. An increase in complexity results in the need to handle the negative partial products correctly. Replacing the Booth encoders with an additional level of 4:2 adders would remove the negative partial products. Our observation is that an increase in speed and reduction in complexity can be obtained with little or no increase in area.[7]

SPIM uses full static master slave latches for testing purposes. These latches are quite large, accounting for 27% of the array size. In addition they are slow, requiring 25% of the cycle time. Since the SPIM architecture has been proven, these latches are not required on future versions. One obvious choice is simply to replace the full static master slave version with dynamic latches. Another option is to split the master slave latches into two separate half latches and incorporate them into the CSA cells. This would reduce area and increase speed. A still more efficient structure, is the use of single phase dynamic latches. The balanced pipe nature of the multiplier makes the use of single phase latches possible. Since only half as many latches are required in the

[7]Replacing the Booth encoders and select muxes with an additional level of 4:2 compressors is a viable alternative on more conventional, i.e. non-piped and non iterative, trees as well. The non-pipelined speed gain depends upon the relative speed of the Booth encode plus Booth select mux vs. the delay through one 4:2 compressor and a NAND gate.

pipe, single phase dynamic latches would reduce the cycle time and decrease latch area.

Research on piped 4:2 trees and accumulators has continued. A test circuit consisting of a new clock generator and an improved 4:2 adder has been fabricated in a 0.8μm CMOS technology. Preliminary test results have demonstrated performance in the range of 400MHz.

VII. Conclusion

SPIM was fabricated in a 1.6μm CMOS process through the DARPA MOSIS fabrication service. It ran at an internal clock speed of 85MHz at room temperature. The latency for a 64 X 64 bit fractional multiply is under 120ns. In piped mode SPIM can initiate a multiply every 4 cycles (47ns), for a throughput in excess of 20 million multiplies per second. SPIM required an average of 72mA at 85MHz, and only 10mA in standby mode. SPIM contains 41 thousand transistors with a core size of 3.8 X 6.5mm, and an array size of 2.9 X 5.3mm.

The 4:2 adder yields a tree structure which is as efficient and far more regular than a Wallace type tree and is therefore better suited for a VLSI implementation. By using a partial 4:2 tree with a carry save accumulator a multiplier can be built which is both faster and smaller than a comparable conventional array. Future designs implemented in a 0.8μm CMOS technology should be capable of clock speeds approaching 400MHz.

Acknowledgements

The development of SPIM was partially supported by the Defense Advanced Project Research Agency (DARPA) under contracts MDA903-83-C-0335 and N00014-87-K-0828. Fabrication support through MOSIS is also gratefully acknowledged.

References

[1] S. F. Anderson, J. G. Earle, et al., "The IBM system/360 Model 91: Floating-Point Execution Unit", IBM Journal, VOL. 11, NO. 1, pp. 34-53, January 1967.

[2] A. D. Booth, "A Signed Binary Multiplication Technique", Qt. J. Mech. Appl. Math., Vol. 4, Part 2, 1951.

[3] J. F. Cavanagh, "Digital Computer Arithmetic Design and Implementation", McGraw-Hill, 1984.

[4] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, Vol. 34, No. 5, pp. 349-356, March 1965.

[5] B. Elkind, J. Lessert, J. Peterson, and G. Taylor, "A Sub 10 ns Bipolar 64 Bit Integer/Floating Point Processor Implemented on Two Circuits", IEEE Bipolar Circuits and Technology Meeting, pp. 101-104, September 1987.

[6] K. Hwang, "Computer Arithmetic: Principles, Architecture, and Design", New York, John Wiley & Sons, 1979.

[7] P. Y. Lu, et al., "A 30-MFLOP 32b CMOS Floating-Point Processor", IEEE Solid-State Circuits Conference proceedings Vol. XXXI, pp. 28-29, February 1988.

[8] W. McAllister and D. Zuras, "An nMOS 64b Floating Point Chip Set", IEEE Int. Solid-State Circuits conf., pp. 34-35, February 1986.

[9] C. S. Wallace, "A Suggestion for Fast Multipliers", IEEE Trans. Electronic Computers, Vol. EC-13, pp. 14-17, February 1964.

[10] S. Waser, and M. J. Flynn, "Introduction to Arithmetic for Digital Systems Designers", New York, CBS Publishing, 1982.

[11] D. Zuras, and W. McAllister, "Balanced Delay Trees and Combinatorial Division in VLSI", IEEE Journal of Solid-State Circuits, VOL. sc-21, no. 5, October 1986.

11/10/88
SPIM
Santoro and Horowitz

Captions for Tables

Table 1: Truth table for the 4:2 adder.

where:
n is number of inputs (from In1, In2, In3, In4) which = 1
Cin is the input carry from the Cout of the adjacent bit slice
Cout and Carry both have weight 2
Sum has weight 1

NOTES:

* Either Cout or Carry may be "1" for 2 or 3 inputs equal to 1 but NOT both.

  Cout may NOT be a function of the Cin from the adjacent block or a ripple carry may occur.

Table 2: SPIM pipe timing. Numbers indicate which partial products are being reduced. 0 is the least significant bit.

n	Cin	Cout	Carry	Sum
0	0	0	0	0
1	0	0	0	1
2	0	*	*	0
3	0	*	*	1
4	0	1	1	0
0	1	0	0	1
1	1	0	1	0
2	1	*	*	1
3	1	1	1	0
4	1	1	1	1

Table 1
Santoro and Horowitz

Action \ Cycle	0	1	2	3	4	5	6	7
Booth Encode	startup 0-15	16-31	32-47	48-63				
A and B block Booth Muxs		0-15	16-31	32-47	48-63			
A Block CSA's			0-7	16-23	32-39	48-55		
B Block CSA's			8-15	24-31	40-47	56-63		
C Block				0-15	16-31	32-47	48-63	
D Block					clear 0-15	16-31	32-47	48-63

Table 2
Santoro and Horowitz

11/10/88
SPIM
Santoro and Horowitz

Figure Captions

Figure 1. Conventional Array Multiplier. Shaded areas represent intermediate partial product flowing down array.

Figure 2. Minimal Iterative Structure using a single row of carry save adders. Black bars represent latches.

Figure 3. A conventional structure (a) has depth proportional to N, while a tree structure (b) has depth proportional to logN.

Figure 4. Block diagram of a 4:2 adder.

Figure 5. A 4:2 adder implemented with two carry save adders.

Figure 6. With the same 4 CSA cells a 4 input partial tree structure with a carry save accumulator (a) will attain almost twice the throughput of a partial piped array (b). In (a) the carry save accumulator is placed under the 4:2 adder.

Figure 7. An 8 input tree constructed from 4:2 adders can reduce 8 partial products per cycle.

Figure 8. Architectural comparison of piped partial tree structure with carry save accumulator vs. conventional partial array.

Figure 9. The SPIM Data Path.

Figure 10. Microphotograph of SPIM.

Figure 11. SPIM clock generator circuit.

Figure 12. Booth encoding vs. additional tree level. The Booth encoders and Booth select muxes (a) can be replaced with an additional level of 4:2 adders and AND gates (b).

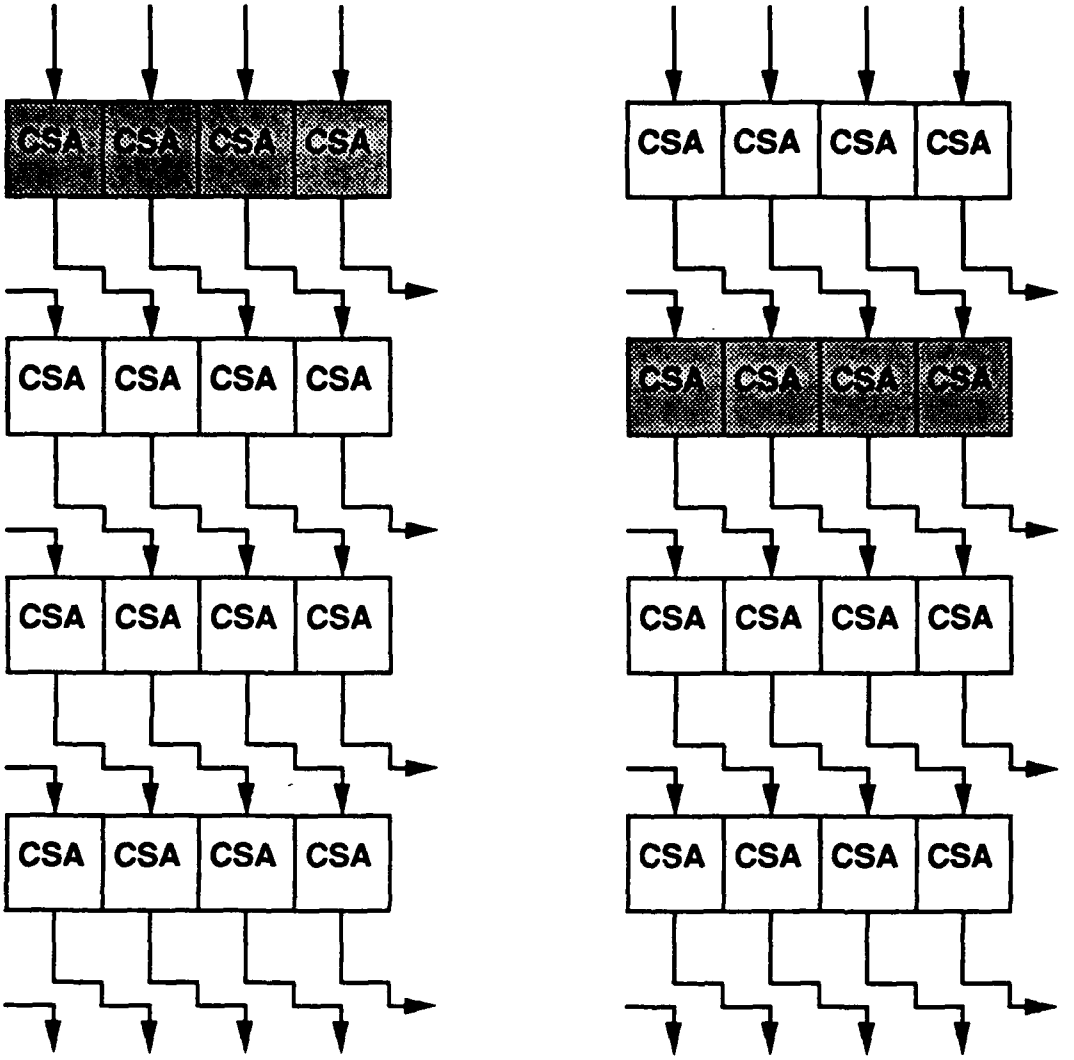

Figure 1
Santoro and Horowitz

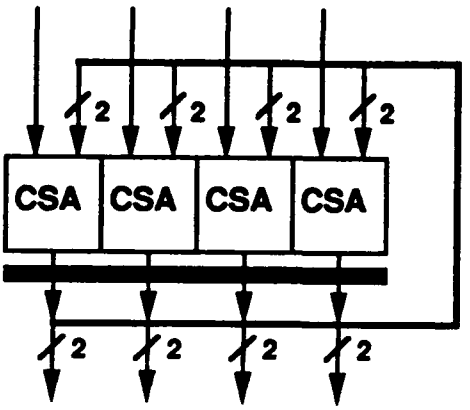

Figure 2
Santoro and Horowitz

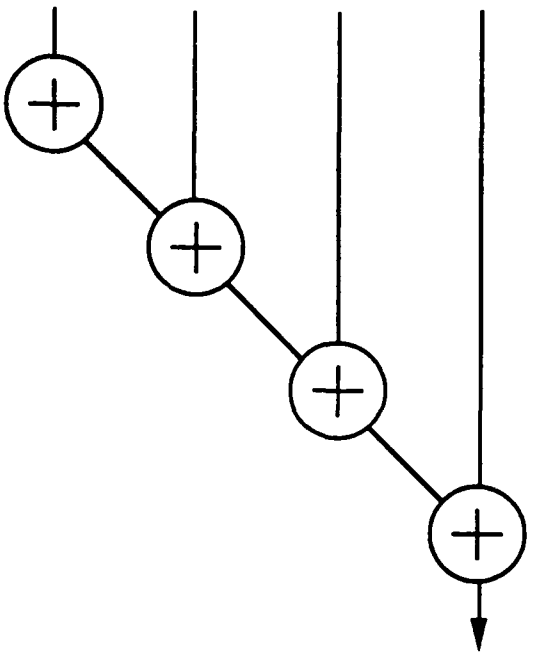

Figure 3a
Santoro and Horowitz

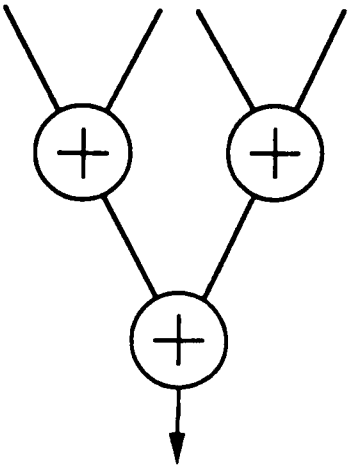

Figure 3b
Santoro and Horowitz

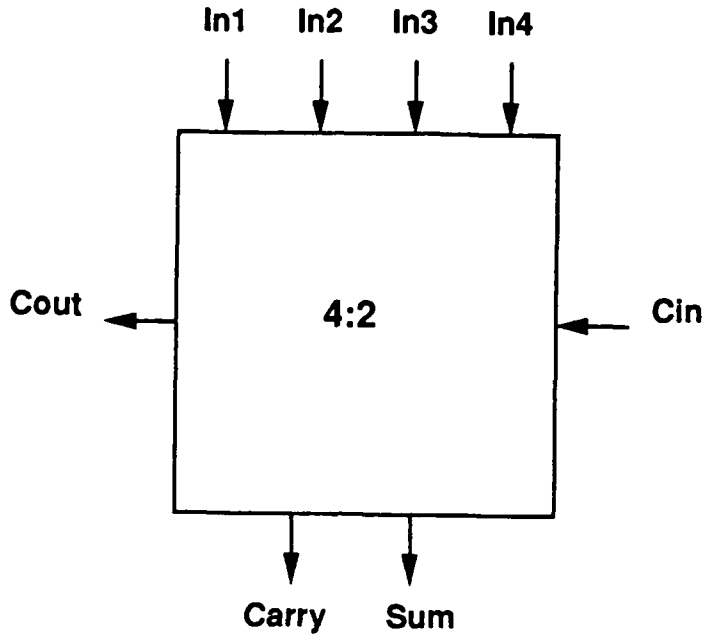

Figure 4
Santoro and Horowitz

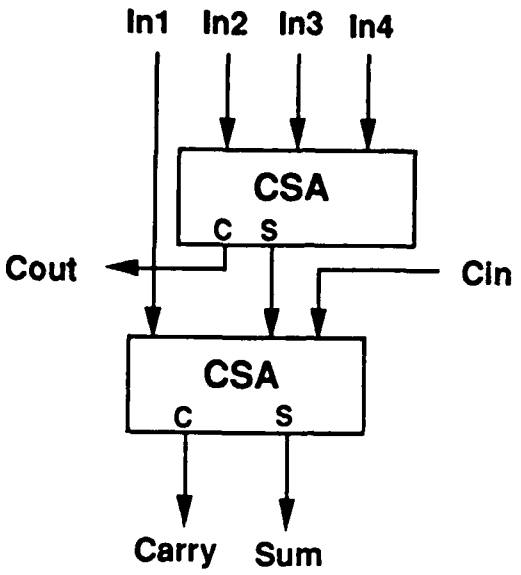

Figure 5
Santoro and Horowitz

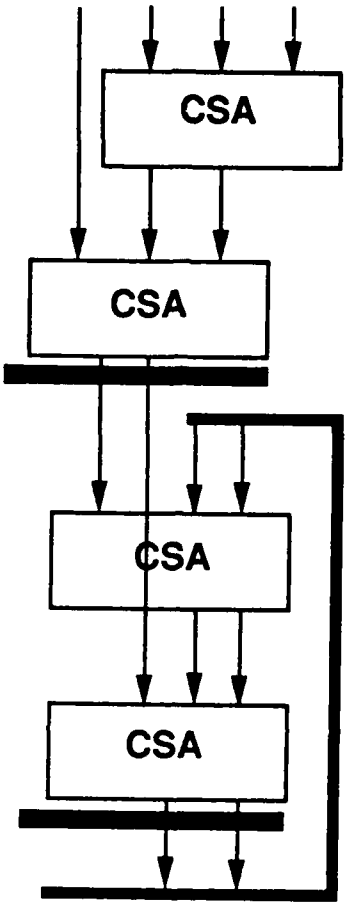

Figure 6a
Santoro and Horowitz

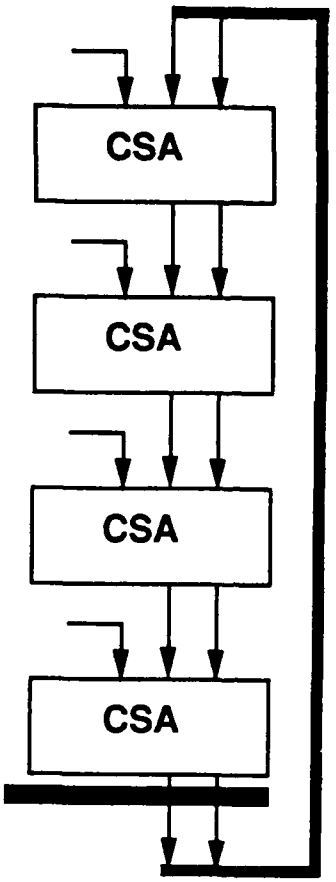

Figure 6b
Santoro and Horowitz

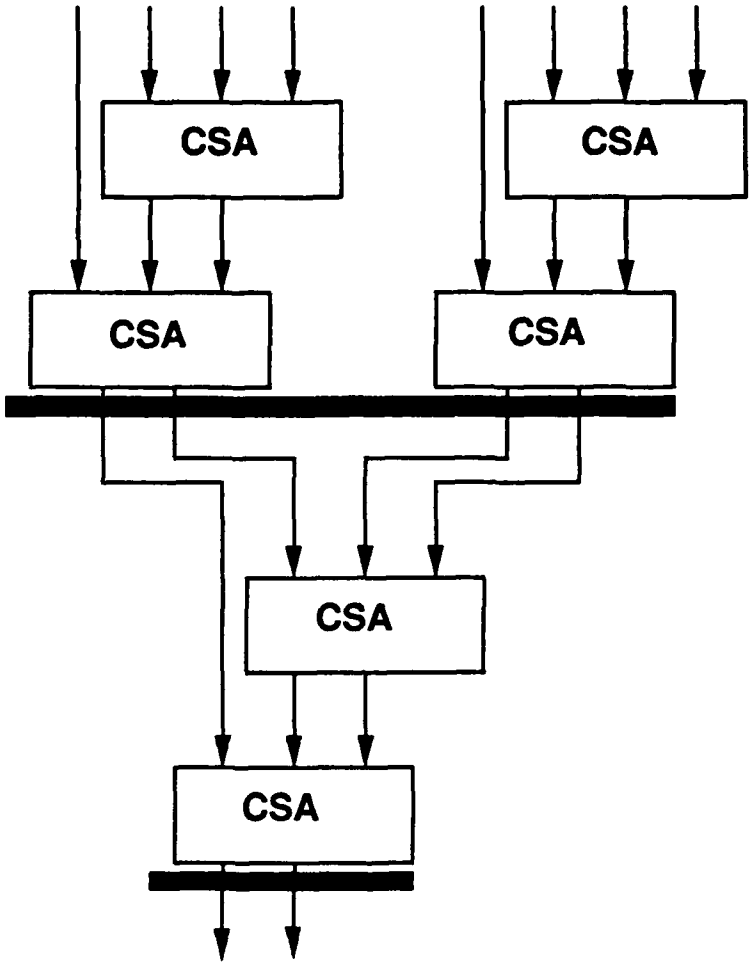

Figure 7
Santoro and Horowitz

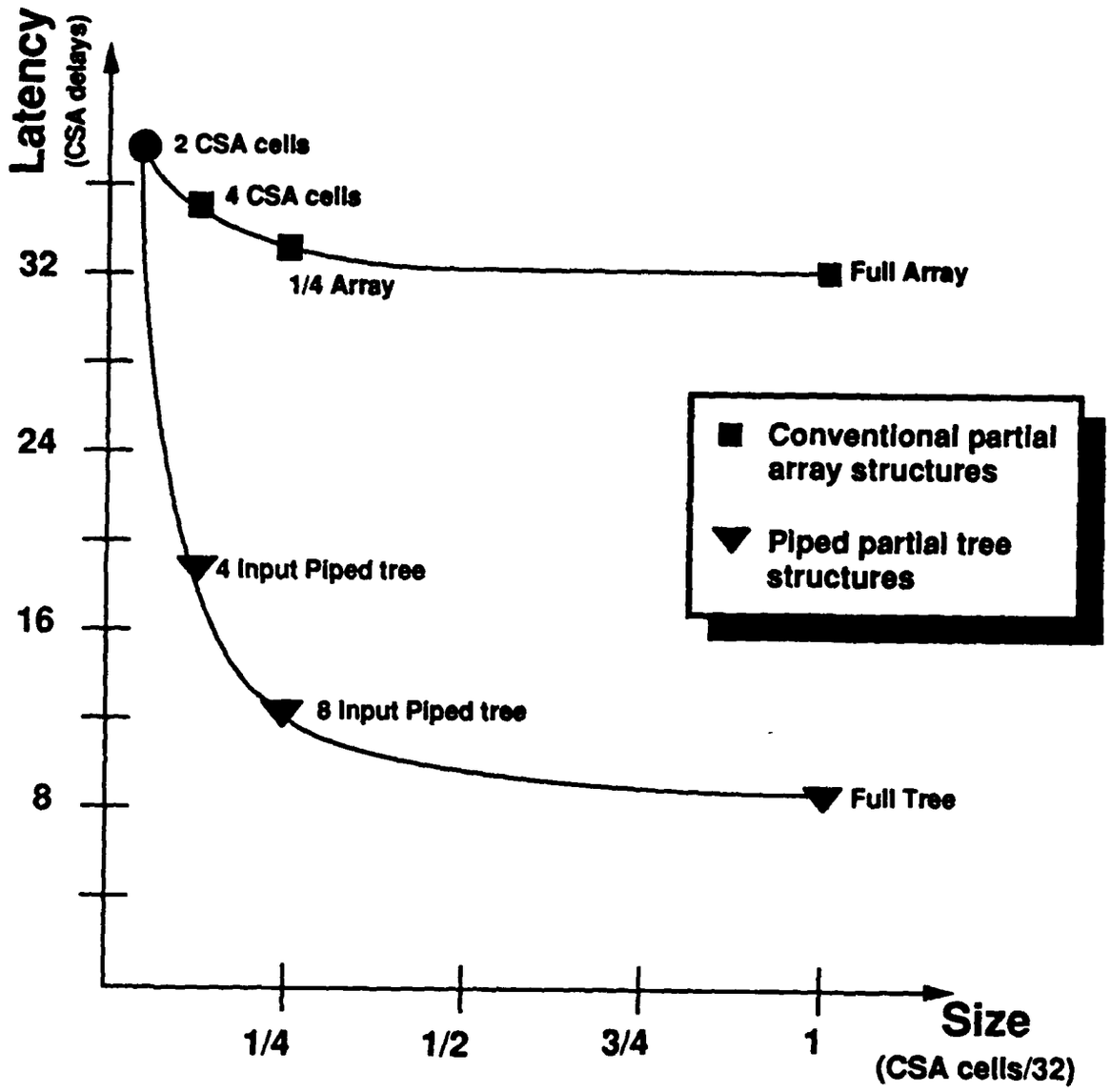

Figure 8
Santoro and Horowitz

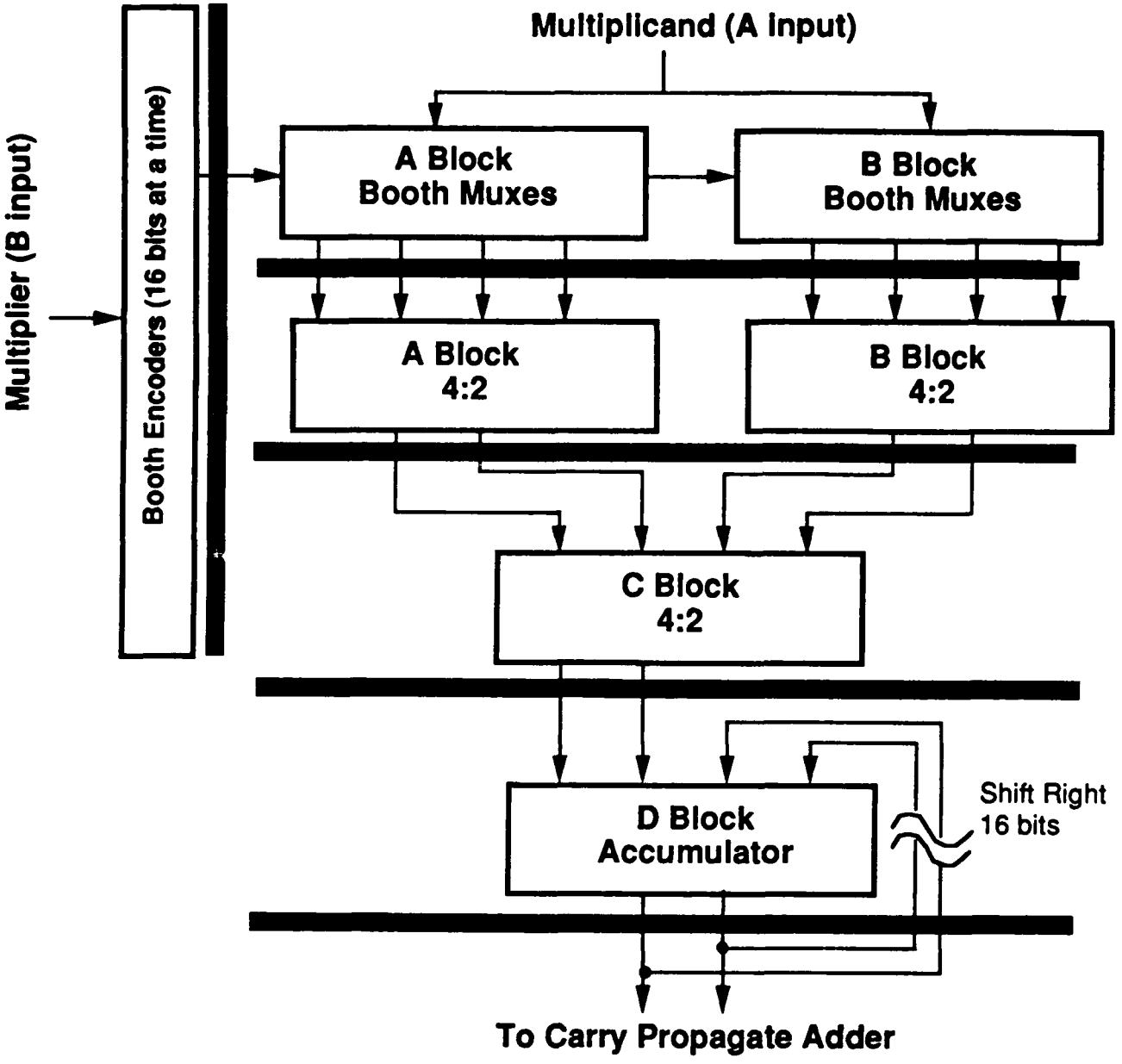

Figure 9
Santoro and Horowitz

A input latches
A Block
B Block
C Block
D Block
CPAdder and Output bus drivers
Control and Clock Buffers
Booth Encoders
B input latches

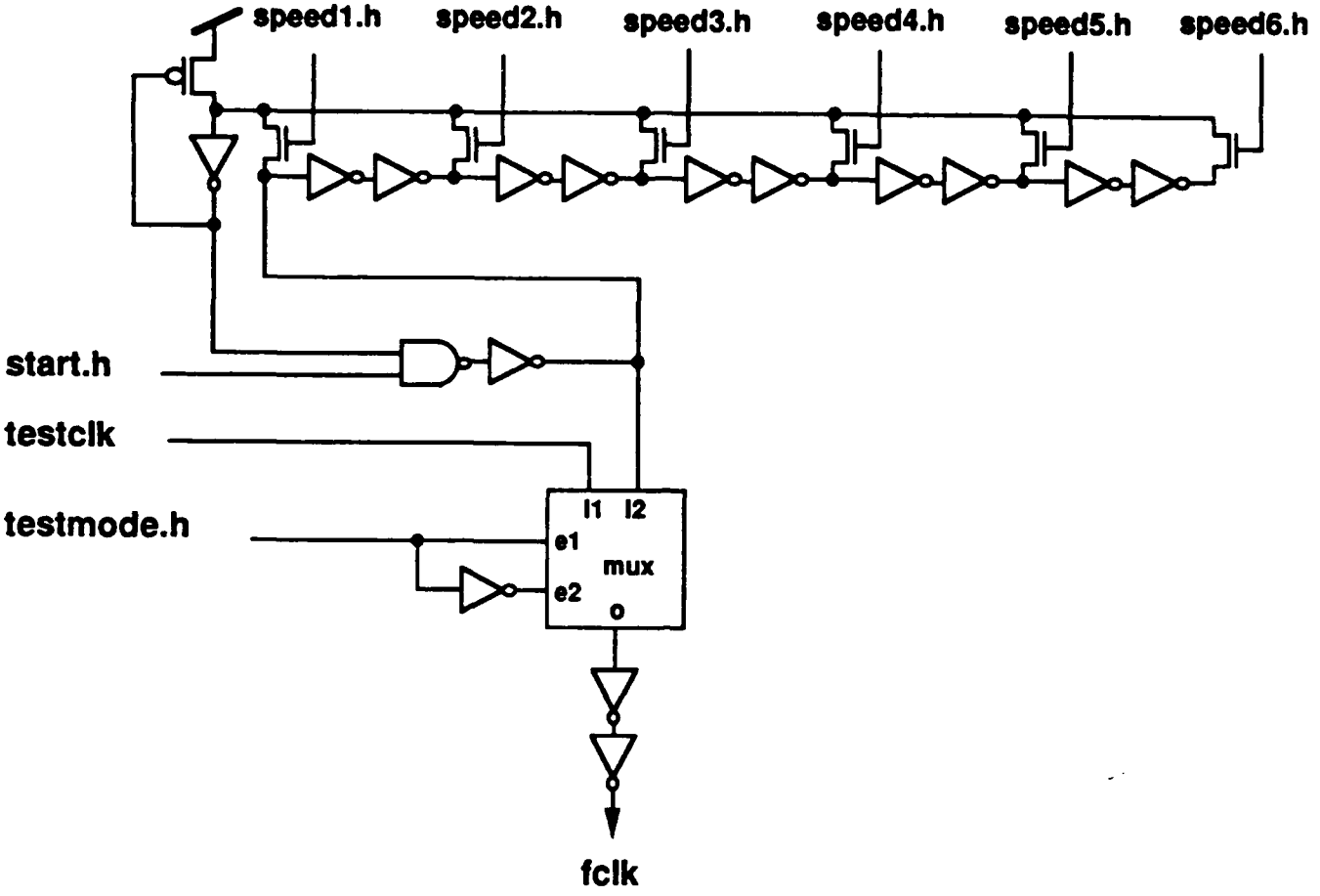

Figure 11
Santoro and Horowitz

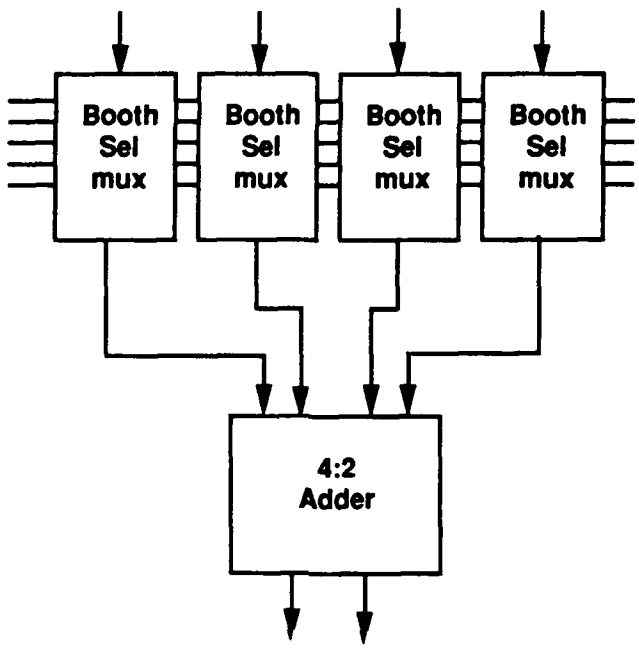

Figure 12a
Santoro and Horowitz

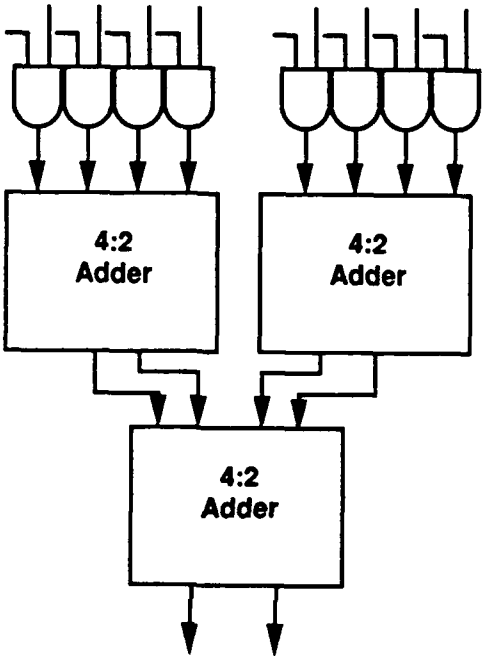

Figure 12b
Santoro and Horowitz